DW02_mult

Multiplier

Version, STAR, and myDesignWare Subscriptions: IP Directory
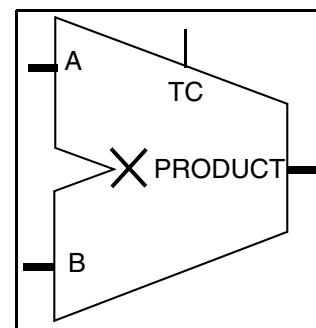
Features and Benefits

- Parameterized word length
- Unsigned and signed (two's-complement) data operation

Description

DW02_mult is a multiplier that multiplies the operand A by B to produce the output, PRODUCT.

The control signal TC determines whether the input and output data is interpreted as unsigned (TC = 0) or signed (TC = 1) numbers.

Table 1-1   Pin Description

Pin Name	Width	Direction	Function
A	*A_width* bits	Input	Multiplier
B	*B_width* bits	Input	Multiplicand
TC	1 bit	Input	Two's complement control ■ 0 = Unsigned ■ 1 = Signed
PRODUCT	*A_width* + *B_width* bits	Output	Product A × B

Table 1-2   Parameter Description

Parameter	Values	Description
A_width	≥ 1	Word length of A
B_width	≥ 1	Word length of B

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
pparch[a]	Delay-optimized flexible architecture	DesignWare
apparch[b]	Area-optimized flexible architecture	DesignWare

a. The 'pparch' (optimized for delay) and 'apparch' (optimized for area) implementations are dynamically generated to best meet your constraints. The 'pparch' and 'apparch' implementations can generate a variety of multiplier architectures including Radix-2 non-Booth, Radix-4 non-Booth, Radix-4 Booth recoded and Radix-8 Booth recoded. Generation of 'pparch' and 'apparch' implementations automatically detects which input would be best suited to be considered the multiplier as opposed to the multiplicand (for Booth recoding). The 'pparch' and 'apparch' implementations are generated making use of any special arithmetic technology cells that are found to be available in your target technology library. The `dc_shell` command, `set_dp_smartgen_options`, can be used to force specific multiplier architectures. For more information on forcing generated arithmetic architectures, use 'man set_dp_smartgen_options' (in dc_shell) to get a listing of the command options.

Table 1-4 Simulation Models

Model	Function
DW02.DW02_MULT_CFG_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW02_mult_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW02_mult.v	Verilog simulation model source code

Table 1-5 Functional Description

TC	A	B	PRODUCT
0	A (unsigned)	B (unsigned)	$A \times B$ (unsigned)
1	A (two's complement)	B (two's complement)	$A \times B$ (two's complement)

Related Topics

- Math – Arithmetic Overview

- DesignWare Building Block IP User Guide

HDL Usage Through Operator Inferencing - VHDL

```vhdl
library IEEE, DWARE;
use DWARE.DW_foundation_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity DW02_mult_oper is
  generic(wordlength1, wordlength2: integer := 8);
  port(in1     : in std_logic_vector(wordlength1-1 downto 0);
       in2     : in std_logic_vector(wordlength2-1 downto 0);
       control : in std_logic;
       mult    : out std_logic_vector(wordlength1+wordlength2-1 downto 0) );
end DW02_mult_oper;

architecture func of DW02_mult_oper is
  signal mult_sig : SIGNED(wordlength1+wordlength2-1 downto 0) ;
  signal mult_usig : UNSIGNED(wordlength1+wordlength2-1 downto 0) ;
begin
  mult_sig <= SIGNED(in1) * SIGNED(in2);
  mult_usig <= UNSIGNED(in1) * UNSIGNED(in2);

  process (mult_sig,mult_usig, control)
  begin
    if control = '1' then
      mult <= std_logic_vector(mult_sig);
    else
      mult <= std_logic_vector(mult_usig);
    end if;
  end process;
end func;
```

HDL Usage Through Operator Inferencing - Verilog

```verilog
module DW02_mult_oper (in1, in2, control, product);
  parameter wordlength1 = 8, wordlength2 = 8;
  input [wordlength1-1:0] in1;
  input [wordlength2-1:0] in2;
  input control;

  output [wordlength1+wordlength2-1:0] product;
  wire signed [wordlength1+wordlength2-1:0] product_sig;
  wire [wordlength1+wordlength2-1:0] product_usig;

  assign product_sig = $signed(in1) * $signed(in2);
  assign product_usig = in1 * in2;
  assign product = (control == 1'b1) ? $unsigned(product_sig) : product_usig;
endmodule
```

HDL Usage Through Function Inferencing - Verilog

```verilog
module DW02_mult_func (in1, in2, prod1, prod2);
  parameter func_A_width = 8, func_B_width = 8;

  // Pass the widths to the DWF_mult functions
  parameter A_width = func_A_width, B_width = func_B_width;

  // Please add search_path = search_path + {synopsys_root + "/dw/sim_ver"}
  // to your .synopsys_dc.setup file (for synthesis) and add
  // +incdir+$SYNOPSYS/dw/sim_ver+ to your verilog simulator command line
  // (for simulation).
  `include "DW02_mult_function.inc"

  input [func_A_width-1 : 0] in1;
  input [func_B_width-1 : 0] in2;
  output [func_A_width-1 : 0] prod1, prod2;

  assign prod1 = DWF_mult_tc(in1, in2);
  assign prod2 = DWF_mult_uns(in1, in2);

endmodule
```

HDL Usage Through Component Instantiation - VHDL

```vhdl
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW02_mult_inst is
  generic ( inst_A_width : NATURAL := 8;
            inst_B_width : NATURAL := 8 );
  port ( inst_A  : in std_logic_vector(inst_A_width-1 downto 0);
         inst_B  : in std_logic_vector(inst_B_width-1 downto 0);
         inst_TC : in std_logic;
    PRODUCT_inst : out std_logic_vector(inst_A_width+inst_B_width-1 downto 0)
        );
end DW02_mult_inst;

architecture inst of DW02_mult_inst is
begin

  -- Instance of DW02_mult
  U1 : DW02_mult
    generic map ( A_width => inst_A_width, B_width => inst_B_width )
    port map ( A => inst_A, B => inst_B, TC => inst_TC,
               PRODUCT => PRODUCT_inst );
end inst;

-- pragma translate_off
configuration DW02_mult_inst_cfg_inst of DW02_mult_inst is
  for inst
  end for; -- inst
end DW02_mult_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```verilog
module DW02_mult_inst( inst_A, inst_B, inst_TC, PRODUCT_inst );

  parameter A_width = 8;
  parameter B_width = 8;

  input [A_width-1 : 0] inst_A;
  input [B_width-1 : 0] inst_B;
  input inst_TC;
  output [A_width+B_width-1 : 0] PRODUCT_inst;

  // Instance of DW02_mult
  DW02_mult #(A_width, B_width)
    U1 ( .A(inst_A), .B(inst_B), .TC(inst_TC), .PRODUCT(PRODUCT_inst) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
October 2019	DWBB_201903.5	■ Updated implementations in Table 1-3 on page 2
January 2019	DWBB_201806.5	■ Updated example in "HDL Usage Through Component Instantiation - VHDL" on page 5 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information